APMA 1940X: Topics in Information and Coding Theory
Project 3

Albert Caputo

April 18 2018

1 Synthesizing Music

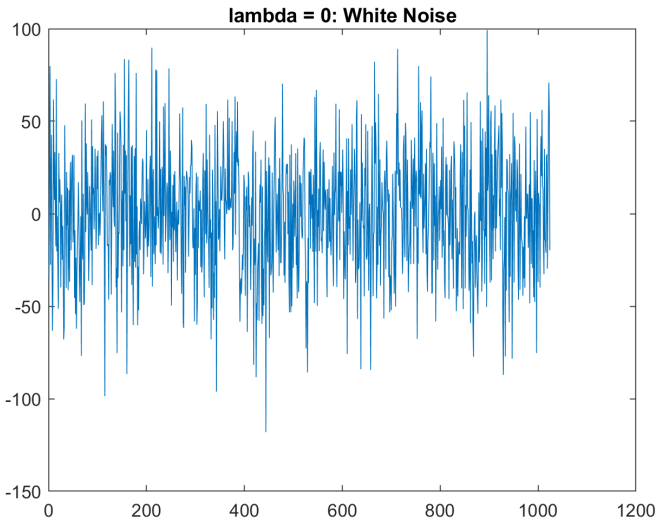
Using the stochastic model for music, we can synthesize music randomly. We begin by synthesizing colored noise and proceed to more structured models as we go along.

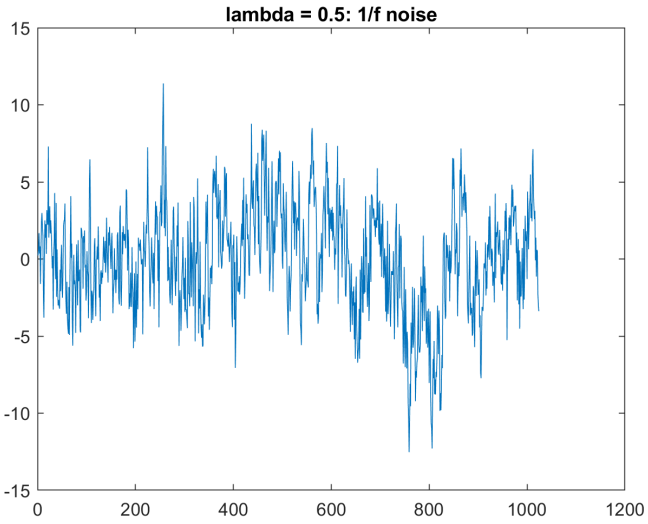
1.1 Synthesizing Colored Noise

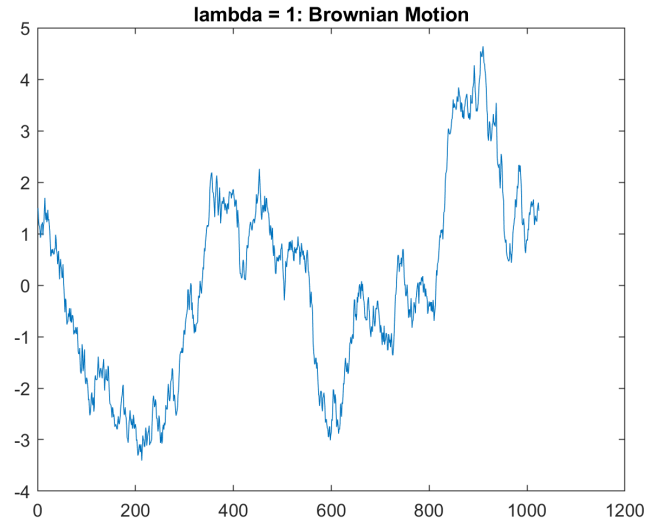
To synthesize colored noise, we first take the real and imaginary parts of the Fourier coefficients of our signal to be random Gaussian variables. That is, for $\hat{s}_k$,

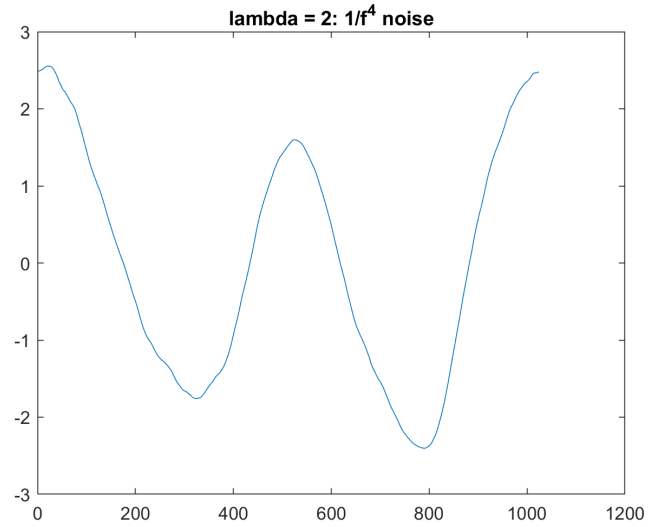
$$\begin{aligned}\hat{s}_k &= x_k + iy_k \\ x_k &\sim \mathcal{N}(0,1) \cdot \frac{1}{\min(k, N-k)^\lambda} \\ y_k &\sim \mathcal{N}(0,1) \cdot \frac{1}{\min(k, N-k)^\lambda}\end{aligned}$$

for some value λ and $k \in \{0, ..., N\}$. We also add the properties $\hat{s}_0 = 0$ and $\hat{s}_{N-k} = \bar{\hat{s}}_k$ to ensure that the Fourier transform is real valued. We then achieve $\vec{s} = FFT(\hat{s})$ where FFT is the fast Fourier transform. We can also ignore the property $\hat{s}_{N-k} = \bar{\hat{s}}_k$ by simply just taking the real part of the Fourier transform. Using the following MATLAB code, we will plot synthesized colored noise with $N = 1024$ and $\lambda = 0, 0.5, 1, 2$:









Increasing λ thus increases the "smoothness" of the output.

1.2 MATLAB Code

```
function [s] = synthColNoise(N,l)
%SYNTHCOLNOISE synthesizes color noise for a power parameter l

x = randn(N,1);
y = randn(N,1);

x(1) = 0;
y(1) = 0;
for k = 2:N-1
        x(k) = x(k)*(1/min(k,N-k)^l);
        y(k) = y(k)*(1/min(k,N-k)^l);
end
s = (1/sqrt(N))*real(fft(x+1i*y));

plot(s)
```

1.3 Synthesizing Notes

To synthesize random notes, we use the model for music as outlined in section 2.3 of *Pattern Theory*. We take the Gaussian distribution of a signal $\vec{s}$ of length N and period p to be

$$\frac{1}{Z}e^{-Q(\vec{s})}$$

where

$$Q(\vec{s}) = \sum_{k=1}^{N-1} \frac{a}{2}(s(k+p) - s(k))^2 + \frac{b}{2}s(k)^2.$$

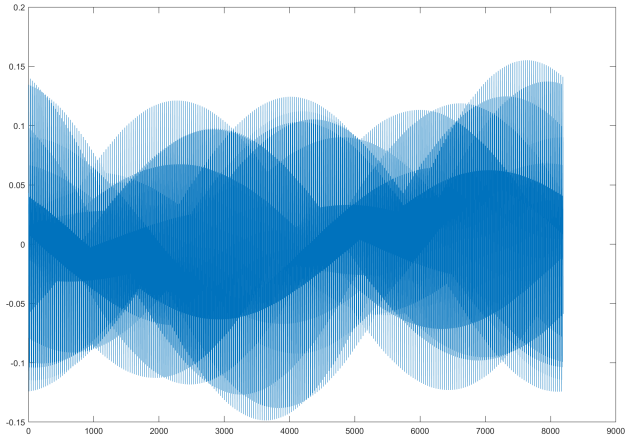
The expected power at frequency l turns out to be

$$\mathbb{E}[|\hat{s}(l)|^2] = \frac{1}{b + 4a\sin^2(\frac{\pi pl}{N})}.$$

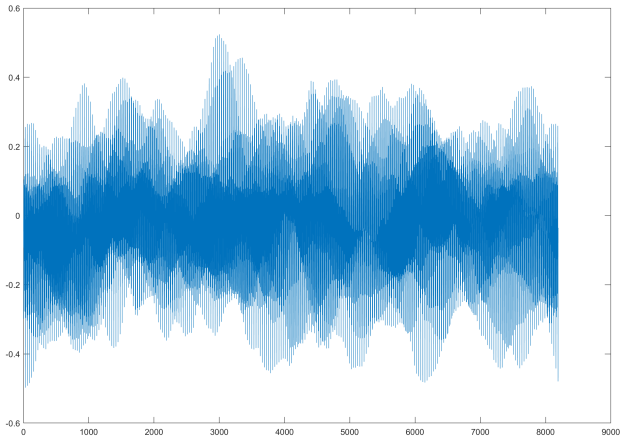
To simulate musical notes in MATLAB, we proceed as follows:

$$\begin{aligned}
\hat{s}_k &= x_k + iy_k \\
x_k &\sim \mathcal{N}(0,1) \cdot \frac{1}{b + 4a\sin^2(\frac{\pi pk}{N})} \\
y_k &\sim \mathcal{N}(0,1) \cdot \frac{1}{b + 4a\sin^2(\frac{\pi pk}{N})} \\
\vec{s} &= \text{Re}(FFT(\hat{s})).
\end{aligned}$$

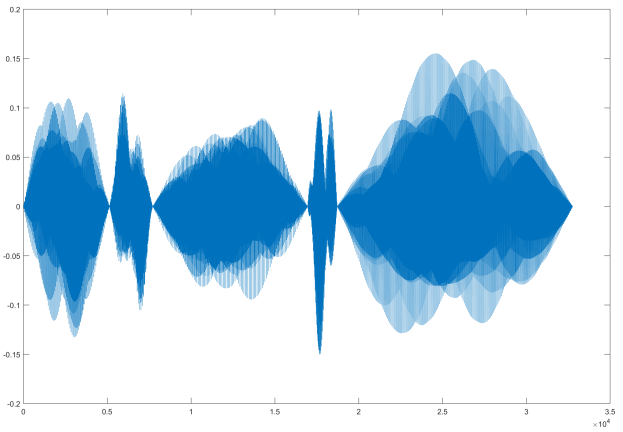
Leaving $b = 1$, we can change the value of a to find notes that sound musical. Setting $a = 100000$, $N = 8192$, and $p = 32$, we achieve the following plot of $\vec{s}$:



We can see that $\vec{s}$ is composed of many different smooth harmonics which come together to sound like a musical note. If we set $a = 100$, we see an output with more noise:



a seems to have an effect on the smoothness of the output similar to λ in the previous section. Let's now try to improve the model using a constant value of a, say $a = 1000$, and choosing random note boundaries using a Poisson Process. Furthermore, we will choose p uniformly at each note boundary. We will synthesize notes in the same way in between note boundaries, but will add an amplitude correction. To do this, we multiply s by $f(x) = -x^2+1,\ x \in [-1, 1]$ discretized over the range of n_{i-1} to n_i at note i. Doing this will give 0 amplitude at the ends of the note and full amplitude in the middle, giving a more realistic note. If we do this such that $N = 2^{15}$, we get a sequence of notes that looks like this:



It does sound like a weird composer randomly choosing note lengths and periods, but this is exactly what we are doing to simulate music.

1.4 MATLAB Code

```
N = 2^15;
a = 10000;
b = 1;

% All Possible Periods
pos_p = [5 8 10 16 20 25 32 40];
notes = poissonProcess(2^15,3000);
ss = zeros(2^15,1);

for i = 2:length(notes)
        p = randsample(pos_p,1,true,ones(length(pos_p),1));
```

```
        s = synthNotes(notes(i)-notes(i-1)+1,a,b,p);
        N = length(s);
        x = -(1-1/N):(2/N):1;
        f = (-1*x.^2+1)';
        % Apply Power Function
        s = f.*s;
        ss(notes(i-1):notes(i)) = s;

end
sound(ss)

function [notes] = poissonProcess(n,lambda)
%POISSONPROCESS creates takes as input a total length n, an expected length
%lambda, and outputs note boundaries

notes = zeros(n,1);
t0 = lambda;
t = -log(rand(1,n))*t0;
notes(1) = 1;
for i = 2:n-1
        if rand(1) > t(i)
                notes(i) = 1;
        else
                notes(i) = 0;
        end
end

notes(n) = 1;
notes = find(notes == 1);
```

2 Parsing Beran's Oboe Cadenza

We are given an audio file containing five seconds of an oboe piece. Our goal is to figure out where the different note boundaries are most likely to be as well as the frequencies of the notes themselves using dynamic programming. The file is represented as samples of the sound wave taken at 8,000 Hertz over five seconds to yield 40,000 total numbers. We know for certain that over the course of the interval, the oboe plays 22 notes, however in our model we will assume we don't know K. Thus, we seek to find

$$0 = n_0 < n_1 < ... < n_K = 40000$$

where the kth note is played in the interval $[n_{k-1}, n_k]$. We will approximate the discrete period of the kth note to be the integer p_k and model the note in this interval by the Gaussian distribution

$$p(s|_{[n_{k-1},n_k]}) = \frac{1}{Z} e^{-\sum_{n_{k-1}}^{n_k - p_k} \left(s(n+p_k)-s(n)\right)^2}$$

where s contains the sound sample and Z is a normalization constant. The entire distribution is therefore

$$p(s, \vec{n}, \vec{p}) = \frac{1}{Z} e^{-\sum_{k=1}^{K} \sum_{n_{k-1}}^{n_k - p_k} \left(s(n+p_k)-s(n)\right)^2}$$

where K is the total number of notes. We seek to maximize $p(s, \vec{n}, \vec{p})$ over $\vec{n}$ and $\vec{p}$. To do so, we can maximize the log-likelihood (or equivalently, minimize the negative log-likelihood), which depends only on the exponent. Furthermore, we can approximate the exponent using the look-up table C where

$$C(n,p) = \sum_{k=1}^{n} s(k)s(k+p)$$

and

$$\sum_{k=1}^{K} \sum_{n_{k-1}}^{n_k - p_k} \left(\left(s(n+p_k) - s(n)\right)^2 \approx 2 \sum_{n=1}^{40000} s(n)^2 - 2 \sum_{k=1}^{K} \left(C(n_k, p_k) - C(n_{k-1}, p_k)\right).$$

If we seek to minimize the above quantity, this is the same as solving

$$\max_{\vec{n},\vec{p}} \sum_{k=1}^{K} \left(C(n_k, p_k) - C(n_{k-1}, p_k)\right) \tag{1}$$

since the first sum is constant. Since we assume K is unknown, we will put an exponential prior on K where $\mathbb{P}(K = k) = \frac{1}{Z}e^{-ak}$. We also assume $5 \leq p \leq 40$ where p must divide 8000. It makes sense that increasing a would segment the music into too few notes while decreasing a would do the opposite. If we set $a = 1/22$, we get that $K = 22$ on average.

To solve (1.1) using dynamic programming, we must iterate over all n, solving for the best possible previous note break and period. Our forward propagation steps are as follows:

1. Define

$$M_1(n_2, p_2) = \max_{n_1;p_1} C(n_2, p_2) - C(n_1, p_2) + C(n_1, p_1) - C(1, p_1)$$
$$R_1(n_2, p_2) = \operatorname*{argmax}_{n_1;p_1} C(n_2, p_2) - C(n_1, p_2) + C(n_1, p_1) - C(1, p_1)$$

   and for $k \in \{3, ..., K-1\}$:

$$M_{k-1}(n_k, p_k) = \max_{n_{k-1};p_{k-1}} C(n_k, p_k) - C(n_{k-1}, p_k) + M_{k-2}(n_{k-1}, p_{k-1})$$
$$R_{k-1}(n_k, p_k) = \operatorname*{argmax}_{n_{k-1};p_{k-1}} C(n_k, p_k) - C(n_{k-1}, p_k) + M_{k-2}(n_{k-1}, p_{k-1})$$

2. For each n from 1 to 40000, solve for the best possible $n_{k-1} \in \{1, ..., n_k - p_k\}$ and $p_{k-1} \in \{5, 8, 10, 16, 20, 25, 32, 40\}$, storing in R_{k-1} for each possible p_k.

We then back propagate:

1. Let

$$M_{K-1}(p_K) = \max_{n_{K-1},p_{K-1}} C(40000, p_K) - C(n_{K-1}, p_K) + M_{K-2}(n_{K-1}, p_{K-1})$$
$$R_{K-1}(n_K, p_K) = \operatorname*{argmax}_{n_{K-1},p_{K-1}} C(n_K, p_K) - C(n_{K-1}, p_K) + M_{K-2}(n_{K-1}, p_{K-1})$$

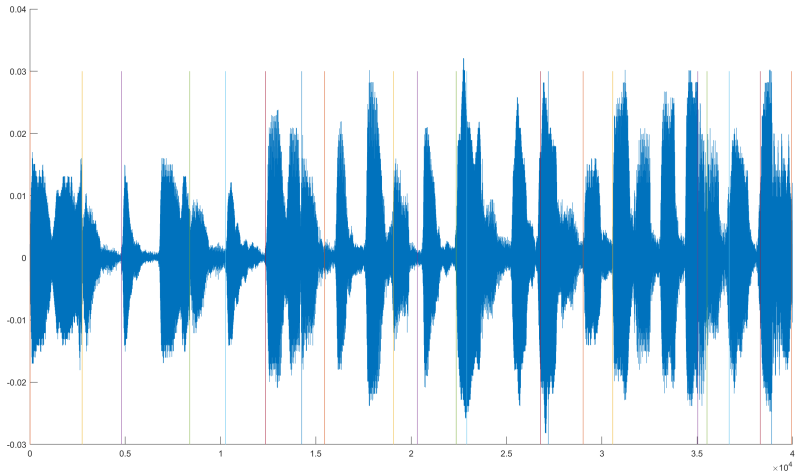
2. Let $n_K = 40000$ and $p_K = \operatorname*{argmax}_{p_K} M_{K-1}(p_K)$.

3. For $i = K-1, ..., 1$

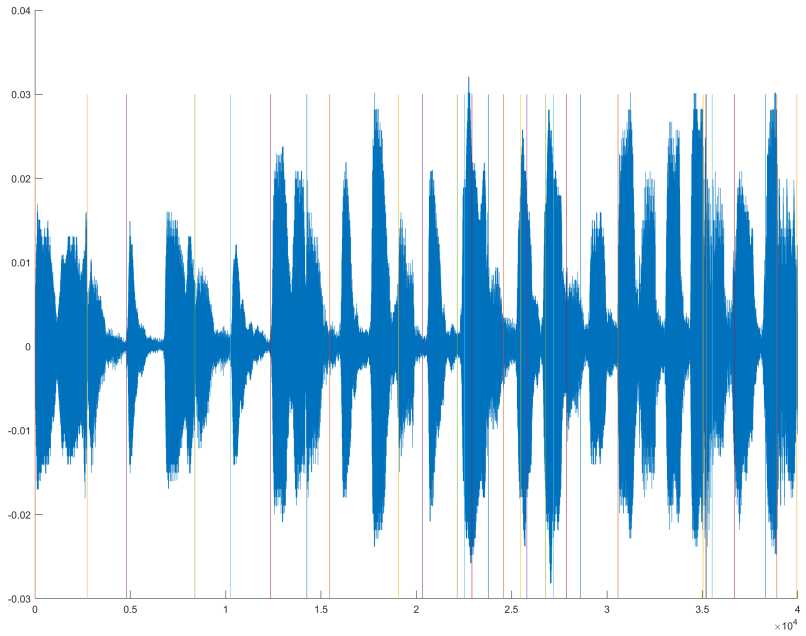
$$n_i, p_i = R_i(n_{i+1}, p_{i+1})$$

4. $n_0 = 1$.

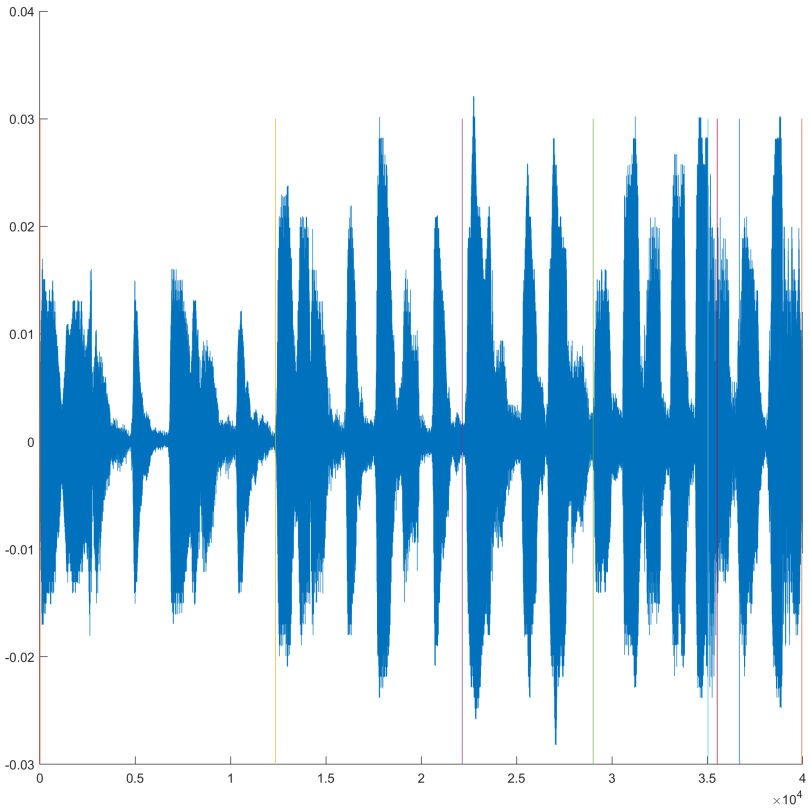
Note that to save memory and speed up run time, we will compress s to s^* such that we take 10 time steps per iteration, leaving $|s^*| = 4000$. Here is a plot of segmentation for $K = 22$:



If we decrease a, we get $K = 30$, for instance, which yields too many segmentations:



Increasing a gives us $K = 8$, which yields too few segmenations:



2.1 MATLAB Code

```
function [C] = calcC(s)
% Takes as input an audio file and outputs the lookup table c
N = length(s)-40;
```

```
p = [5 8 10 16 20 25 32 40];
C = zeros(N,40);

for j = p
        C(1,p) = s(1)*s(1+j);
end

for j = p
        for i = 2:N
                C(i,j) = C(i-1,j) + s(i)*s(i+j);
        end
end

function [M,R,P,N,Ps] = dynamicScore(C,a)
%DYNAMICSCORE takes as input the lookup table C and calculates the most
%likely score
K = 22;
L = length(C);
K = ceil(exprnd(1/a));
M = zeros(L,8,K-1);
R = ones(L,8,K-1);
P = zeros(L,8,K-1);
N = zeros(K,1);
Ps = zeros(K,1);
T = length(C);

p = [5 8 10 16 20 25 32 40];

% Forward Prop
% First step, only look back to 1:n-40 to account for period
for n = 5:L
        for j = 1:8
                h = C(n,p(j)) - C(1:n-4,p(j)) + C(1:n-4,p) - C(1,p);
                [M(n,j,1),idx] = max(h(:));
                [R(n,j,1),P(n,j,1)] = ind2sub(size(h),idx);
        end
end

% Steps 2:K-2
for k = 2:K-2
        for n = 5:L
                for j = 1:8
                        h = (C(n,p(j)) - C(1:n-4,p(j)))+M(1:n-4,:,k-1);
                        [M(n,j,k),idx] = max(h(:));
                        [R(n,j,k),P(n,j,k)] = ind2sub(size(h),idx);
                end
        end
end

% Step K-1
for j = 1:8
        h = C(T,p(j)) - C(1:T,p(j)) + M(1:T,:,K-2);
        [M(T,j,K-1),idx] = max(h(:));
        [R(T,j,K-1),P(T,j,K-1)] = ind2sub(size(h),idx);
end

[~,ind] = max(M(T,:,K-1));
N(K) = T;
Ps(K) = ind;

for k = fliplr(2:K-1)
```

```
        N(k) = R(N(k+1),Ps(k+1),k);
        Ps(k) = P(N(k+1),Ps(k+1),k);
end

Ps(1) = P(N(2),Ps(2),k);
N(1) = 1;

% Upload entire audio file
s = audioread('fivesec.flac');
C = calcC(s);
C = C(1:10:end,:);

[~,~,~,N,Ps] = dynamicScore(C,1/22);
p = [5 8 10 16 20 25 32 40];

hold on
plot(s)
for i = N
        plot([10*i 10*i],[-0.03,0.03])
end
hold off
```

3 Baseball Streaks Expectation Maximization

We are tasked with finding hidden "streaks" in a baseball player's boxscore. A running streak indicates multiple games in a row with at least one run. The longest recorded (hitting) streak is held by Joe DiMaggio of the New York Yankees in 1941. We will use his record from that year. The model is as follows:

Let $x_i = 1$ if Joe got a hit in game i and $x_i = 0$ if he did not for all N games in the 1941 season. We will define the hidden Markov variable y such that $y_i = 1$ if Joe is on a streak and $y_i = 0$ if he is not. y will have some probability λ_c of continuing (streak or non-streak) and we will denote the probability of getting a run while on a streak by multiplying his batting average $b = 0.325$ by some constant λ_1 if he is on a streak. Otherwise, we will multiply his batting average by some other constant λ_0. Hence, for N games,

$$P(\hat{x}, y|\lambda) = \prod_{i=1}^{N-1} \left(\lambda_c \mathbb{1}_{y_{i+1}=y_i} + (1-\lambda_c)\mathbb{1}_{y_{i+1}\neq y_i}\right) \prod_{j=1}^{N} \Big(x_j\big(\lambda_1 b y_j + \lambda_0 b(1-y_j)\big) + (1-x_j)\big((1-b\lambda_1)y_j + (1-b\lambda_0)(1-y_j)\big)\Big)$$

which can be re-written as

$$P(\hat{x}, y|\lambda) = \prod_{i=1}^{N-1} \left(\lambda_c \mathbb{1}_{y_{i+1}=y_i} + (1-\lambda_c)\mathbb{1}_{y_{i+1}\neq y_i}\right) \prod_{j=1}^{N} \Big(\mathbb{1}_{y_j=1,x_j=1} b\lambda_1 + \mathbb{1}_{y_j=1,x_j=0}(1-b\lambda_1) + \mathbb{1}_{y_j=0,x_j=1} b\lambda_0 + \mathbb{1}_{y_j=0,x_j=0}(1-b\lambda_0)\Big).$$

We seek to find the best possible values of $\lambda_c, \lambda_0, \lambda_1$ using the expectation maximization (EM) algorithm:

1. Initilialize λ^0.
2. Expectation step: compute iterations $k = \{1, ..., K\}$ for large enough K

$$Q(\lambda^{k+l}, \lambda^k) = \underset{\lambda_{k+1}}{\text{argmax}} \ \mathbb{E}^{\lambda^k}_{P_{Y|X=x}}[\log\big(P(\hat{x}, y|\lambda^{k+1})\big)]$$

3. Choose λ^K to be the best choice for λ.

This algorithm converges to a local maximum as shown in the proof in *Pattern Theory*. Note that in step 2, we are computing expectation w.r.t. the probability $P(y|\hat{x}, \lambda_k)$. To find the maximum values of $\log(P(\hat{x}, y|\lambda))$, we can simply

take the derivative w.r.t. each λ and set equal to zero under the concavity of log. Let's now compute $\mathbb{E}[\log(P(\hat{x}, y|\lambda))]$:

$$\begin{aligned}
\mathbb{E}[\log(P(\hat{x}, y|\lambda))] &= \mathbb{E}[\sum_{i=1}^{N-1} \log\big(\lambda_c \mathbb{1}_{y_{i+1}=y_i} + (1-\lambda_c)\mathbb{1}_{y_{i+1}\neq y_i}\big) \\
&\quad + \sum_{j=1}^{N} \log\big(\mathbb{1}_{y_j=1,j_i=1} b\lambda_1 + \mathbb{1}_{y_j=1,x_j=0}(1-b\lambda_1) + \mathbb{1}_{y_j=0,x_j=1} b\lambda_0 + \mathbb{1}_{y_j=0,x_j=0}(1-b\lambda_0)\big)] \\
&= \log(\lambda_c)\mathbb{E}[\sum_{i=1}^{N-1} \mathbb{1}_{y_{i+1}=y_i}] + \log(1-\lambda_c)\mathbb{E}[\sum_{i=1}^{N-1} \mathbb{1}_{y_{i+1}\neq y_i}] \\
&\quad + \log(b\lambda_1)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}] + \log(1-b\lambda_1)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_j=0}] \\
&\quad + \log(b\lambda_0)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=0,x_i=1}] + \log(1-b\lambda_0)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=0,x_j=0}]
\end{aligned}$$

Now, we take $\frac{\partial}{\partial\lambda}\mathbb{E}[\log(P(\hat{x}, y|\lambda))]$ for all λ:

First, λ_c:

$$\begin{aligned}
\frac{\partial}{\partial\lambda_c}\mathbb{E}[\log(P(\hat{x}, y|\lambda))] &= \frac{\partial}{\partial\lambda_c}\Big(\log(\lambda_c)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_{i+1}=y_i}] + \log(1-\lambda_c)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_{i+1}\neq y_i}]\Big) \\
&= \frac{\mathbb{E}[\sum_{i=1}^{N-1} \mathbb{1}_{y_{i+1}=y_i}]}{\lambda_c} - \frac{\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_{i+1}\neq y_i}]}{1-\lambda_c} \\
&= \frac{\mathbb{E}[\sum_{i=1}^{N-1} \mathbb{1}_{y_{i+1}=y_i}]}{\lambda_c} - \frac{N-1-\mathbb{E}[\sum_{i=1}^{N-1} \mathbb{1}_{y_{i+1}=y_i}]}{1-\lambda_c} = 0 \\
\lambda_c &= \frac{\mathbb{E}[\sum_{i=1}^{N-1} \mathbb{1}_{y_{i+1}=y_i}]}{N-1}.
\end{aligned}$$

Next, λ_1:

$$\begin{aligned}
\frac{\partial}{\partial\lambda_1}\mathbb{E}[\log(P(\hat{x}, y|\lambda))] &= \frac{\partial}{\partial\lambda_1}\Big(\log(b\lambda_1)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}] + \log(1-b\lambda_1)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_j=0}]\Big) \\
&= \frac{\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}]}{\lambda_1} - \frac{b\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=0}]}{1-b\lambda_1} \\
&= (1-b\lambda_1)\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}] - b\lambda_1\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=0}] \\
&= \mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}] - b\lambda_1\big(\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}] + \mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=0}]\big) \\
&= \mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}] - b\lambda_1\big(\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1}]\big) = 0 \\
\lambda_1 &= \frac{\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}]}{b\cdot\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1}]}.
\end{aligned}$$

By similarity for λ_0:

$$\begin{aligned}
\lambda_0 &= \frac{\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=0,x_i=1}]}{b\cdot\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=0}]} \\
&= \frac{\sum_i^N x_i - \mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{y_j=1,x_i=1}]}{b\big(N-\sum_{i=1}^{N} \mathbb{1}_{y_j=1}\big)}.
\end{aligned}$$

These λ's make a lot of sense! λ_c is the expected ratio of continuations, λ_1 is the expected ratio of streak games with runs to all streak games, and λ_0 is the expected ratio of non-streak games with runs to all streak games. These are exactly how we defined them. We only need to compute three expectations for each EM iteration which we can do using dynamic

programming.

To achieve these values we must compute $P(y|\hat{x}, \lambda)$. Define the following functions as the probabilities of continuing a streak and getting a run on a streak vs. non-streak respectively:

$$a(y_{i+1}, y_i|\lambda_c) = \lambda_c \mathbb{1}_{y_{i+1}=y_i} + (1-\lambda_c)\mathbb{1}_{y_{i+1}\neq y_i}$$
$$f(x_j, y_j|\lambda_0, \lambda_1) = \mathbb{1}_{y_j=1,j_i=1} b\lambda_1 + \mathbb{1}_{y_j=1,x_j=0}(1-b\lambda_1) + \mathbb{1}_{y_j=0,x_j=1} b\lambda_0 + \mathbb{1}_{y_j=0,x_j=0}(1-b\lambda_0)$$

To compute $P(y|\hat{x}, \lambda)$, we first forward propagate using a numerical stability protocol:

$$\begin{aligned}
\text{Temp}(y_2) &= \sum_{y_1} a(y_2, y_1|\cdot) f(y_1, x_1|\cdot) \\
S_1 &= \sum_{y_2} \text{Temp}(y_2) \\
T_1(y_2) &= \frac{\text{Temp}(y_2)}{S_1} \\
\text{Temp}(y_3) &= \sum_{y_2} a(y_3, y_2|\cdot) f(y_2, x_2|\cdot) T_1(y_2) \\
S_2 &= \sum_{y_3} \text{Temp}(y_3) \\
T_2(y_3) &= \frac{\text{Temp}(y_3)}{S_2} \\
&\vdots \\
\text{Temp}(y_N) &= \sum_{y_{N-1}} a(y_N, y_{N-1}|\cdot) f(y_{N-1}, x_{N-1}|\cdot) T_{N-2}(y_{N-1}) \\
S_{N-1} &= \sum_{y_N} \text{Temp}(y_N) \\
T_{N-1} &= \frac{\text{Temp}(y_N)}{S_{N-1}} \\
\text{Temp} &= \sum_{y_N} f(x_N, y_N|\cdot) T_{N-1}(y_N) \\
S_N &= \text{Temp} \\
T_N &= 1
\end{aligned}$$

After we complete forward propagation, we can backward propagate to find $P(y|\cdot)$ and the expectations:

$$\begin{aligned}
P(y_N|\cdot) &= f(x_N, y_N|\cdot)\frac{T_{N-1}(y_N)}{S_N} \\
\mathbb{E}[\mathbb{1}_{y_N=1}] &= P(y_N = 1|\cdot) \\
\mathbb{E}[\mathbb{1}_{y_N=1,x_N=1}] &= P(y_N = 1|\cdot)\mathbb{1}_{x_N=1}
\end{aligned}$$

For $i \in \{2, ..., N-1\}$:

$$\begin{aligned}
P(y_i|\cdot) &= \sum_{y_{i+1}} (P(y_{i+1}|\cdot)a(y_{i+1}, y_i)) \frac{f(x_i, y_i) T_{i-1}(y_i)}{\sum_{y_{i+1}} P(y_{i+1}|\cdot) T_i(y_{i+1}) S_i} \\
\mathbb{E}[\mathbb{1}_{y_i=1}] &= P(y_i = 1|\cdot) \\
\mathbb{E}[\mathbb{1}_{y_i=1,x_i=1}] &= P(y_i = 1|\cdot)\mathbb{1}_{x_i=1} \\
\mathbb{E}[\mathbb{1}_{y_{i+1}=y_i}] &= P(y_{i+1} = 0|\cdot)P(y_i = 0|\cdot) + P(y_{i+1} = 1|\cdot)P(y_i = 1|\cdot)
\end{aligned}$$

And lastly,

$$\begin{aligned}
P(y_1|\cdot) &= \sum_{y_2} (P(y_2|\cdot)a(y_2, y_1)) \frac{f(x_1, y_1|\cdot)}{\sum_{y_2} T_1(y_2) S_2} \\
\mathbb{E}[\mathbb{1}_{y_1=1}] &= P(y_1 = 1|\cdot) \\
\mathbb{E}[\mathbb{1}_{y_1=1,x_1=1}] &= P(y_1 = 1|\cdot)\mathbb{1}_{x_1=1} \\
\mathbb{E}[\mathbb{1}_{y_2=y_1}] &= P(y_2 = 0|\cdot)P(y_1 = 0|\cdot) + P(y_2 = 1|\cdot)P(y_1 = 1|\cdot)
\end{aligned}$$

We just have to sum up each of the individual expectations to achieve the total value. During our EM algorithm, we calculate the above expectations using the old value of λ and output the new value of λ as previously calculated for every iteration. Hence, in MATLAB:

```
function [P,c,y1,y11] = computeExpect(x,lc,l0,l1,BA)
%COMPUTEEXPECT calculates expectations at every step of the EM algorithm

N = length(x);

c = 0;
y1 = 0;
y11 = 0;

T = zeros(N,2);
S = zeros(N,1);
P = zeros(N,2);

a = @(y2,y1) lc*(y2 == y1) + (1-lc)*(y2 ~= y1);
f = @(x,y) x.*BA.*(l1.*y + l0.*(1-y)) + (1-x).*((1-l1.*BA).*y+(1-l0.*BA).*(1-y));

Temp = a([0 1],0).*f(x(1),0) + a([0 1],1).*f(x(1),1);
S(1) = sum(Temp);
T(1,:) = Temp/S(1);

% Forward computation
for i = 2:N-1
        %Compute T for each i
        Temp = T(i-1,1)*a([0 1],0)*f(x(i),0) + T(i-1,2)*a([0 1],1)*f(x(i),1);
        S(i) = sum(Temp);
        T(i,:) = Temp/S(i,:);
end
S(N) = T(N-1,1)*f(x(N),0)+T(N-1,2)*f(x(N),1);

% Backward computation:
P(N,:) = f(x(N),[0 1]).*T(N-1,:)/S(N);
y1 = y1 + P(N,2);
y11 = y11 + (x(N) == 1)*P(N,2);

for i = fliplr(2:N-1)
        P(i,:) = (P(i+1,1).*a(0,[0 1])+P(i+1,2).*a(1,[0 1])).*f(x(i),[0 1])...
        .*T(i-1,:)./sum(P(i+1,:).*T(i,:).*S(i));
        c = c + P(i,1)*P(i+1,1) + P(i,2)*P(i+1,2);
        y1 = y1 + P(i,2);
        y11 = y11 + (x(i) == 1)*P(i,2);
end

P(1,:) = (P(2,1).*a(0,[0 1])+P(2,2).*a(1,[0 1])).*f(x(1),[0 1])/sum(P(2,:).*T(1,:)*S(1));

c = c + P(1,1)*P(2,1) + P(1,2)*P(2,2);
y1 = y1 + P(1,2);
y11 = y11 + (x(i) == 1)*P(1,2);


function [Q] = runsEM(x,lc,l0,l1,iter,BA)
%RUNSEM calculates the best possible lambda for iter number of steps

% Init
N = length(x);
Q = zeros(iter,3);
Q(1,:) = [lc l0 l1];
```

```
% Steps
for k = 2:iter
        [~,c,y1,y11] = computeExpect(x,Q(k-1,1),Q(k-1,2),Q(k-1,3),BA);
        Q(k,1) = c/(N-1);
        Q(k,2) = (sum(x)-y11)/(BA*(N-y1));
        Q(k,3) = y11/(BA*y1);
end
end
```

If we initialize $\lambda = [\lambda_c, \lambda_0, \lambda_1] = [0.6, 0.5, 2]$ and iterate EM 300 times, we achieve the following values of $Q(\lambda^{k+1}, \lambda_k)$ for every 30th k:

```
>> Q(1:30:end,:)

ans =

0.6000    0.5000    2.0000
0.5380    0.8671    2.9327
0.5480    0.7566    3.0386
0.5501    0.7273    3.0671
0.5505    0.7199    3.0744
0.5506    0.7180    3.0763
0.5506    0.7175    3.0768
0.5506    0.7173    3.0769
0.5506    0.7173    3.0769
0.5506    0.7173    3.0769
```

Notice that $1/0.325 = 3.0769$, meaning that EM converges λ_1 to the value such that Joe is guaranteed to get a run while on a streak. Furthermore, streaks/non-streaks are most likely to continue with a %55 chance, and Joe is most likely to only have a $0.7173 \cdot 0.325 = 0.2331 = \%23.31$ chance of getting a run while not on streak.